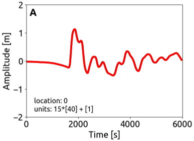
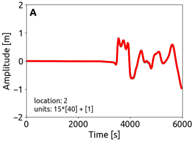
AI for Data-driven Simulations in Physics.

Siddhartha Mishra

Computational and Applied Mathematics Laboratory (CamLab)
Seminar for Applied Mathematics (SAM), D-MATH (and),
ETH AI Center,
ETH Zürich, Switzerland.

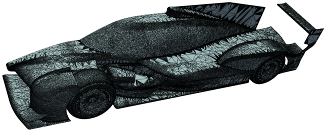
Use Case I: Tsunami Early Warning System@INGV

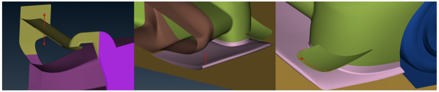




- Task: Predict Wave Height Time Series at different Buoy locations in Real Time
- Basis of Tsunami Evacuation Forecast.

Use Case II: Race Car Design@Dallara



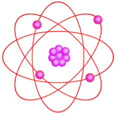


- Optimize Car Design.
- Predict Aerodynamic body force changes by changing specific parts.

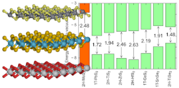
How are these problems solved currently ?

Step I: Mathematical Modeling

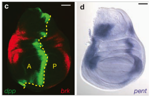
- Model Physical Phenomena with Partial Differential Equations
- PDEs are Language of Nature



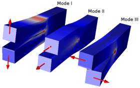
Schrödinger



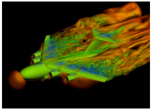
Kohn-Sham



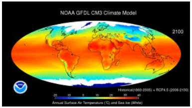
Reaction-Diffusion



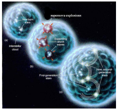
Phase-Field



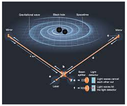
Euler



Navier-Stokes ++



MHD++

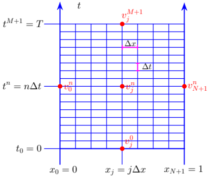


Einstein

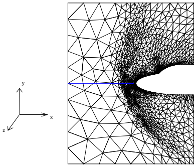
- Immense diversity of Physical processes
- Very wide range of spatio-temporal scales

Step II: Numerical Simulation

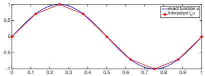
- Not possible to find solution formulas for PDEs.
- Reliance on Numerical Methods to approximate PDEs on computers.



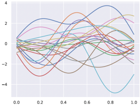
Finite Difference



Finite Volume



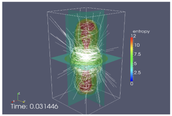
Finite Element

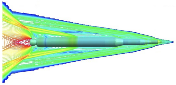


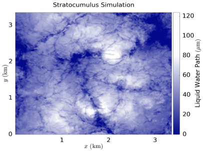
Spectral Method

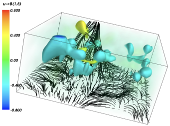
Numerical Methods are very Successful

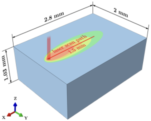
- Including@CAMlab

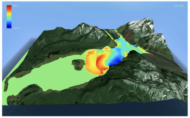




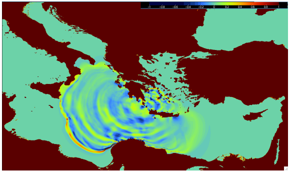




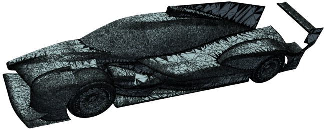
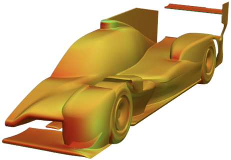




What about the Use Cases ?

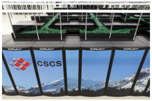



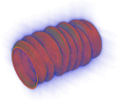
- Tsunami Simulation with Shallow-Water Equations
- Flow past Race car simulation with Navier-Stokes Equations

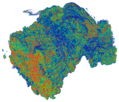



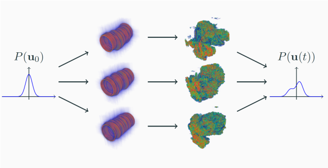
Where is the Catch ?

Issues with Numerical Methods I: Computational Cost



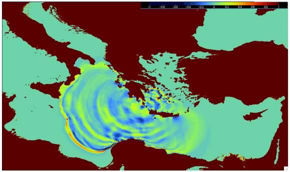




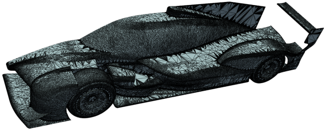


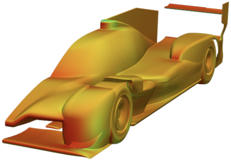
- PDE solvers can be very expensive,
- Many-Query Problems: UQ, Design, Inverse Problems.
- Simulation of Navier-Stokes at 1024^3:
  - With Azeban on Piz Daint.
  - Single Run: 94 GPU hours (4512 CPU hours)
  - Ensemble simulation: 96256 node hours
  - Cost: Approx $500K$ USD.
  - Solve PDEs fast

What about the Examples ?

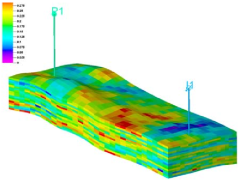


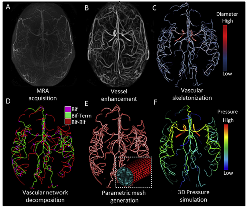
- Single Tsunami Simulation takes > 1 hour !!
- Flow past Race car simulation requires 500 node hours per shape !!

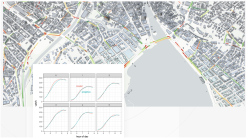


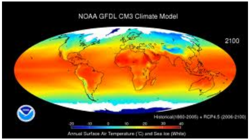


Issues with Numerical Methods II: Unknown Physics



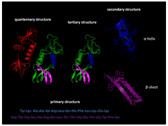
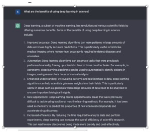




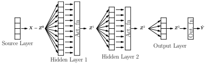


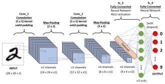
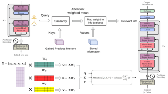
- Missing Physics not just undetermined parameters.
- Manifestation of Sim2Real gap.
- Holds True for most real-world applications.
- Still have Data for the underlying Problem
- Learn PDE Solutions from Data + Physics

The age of AI

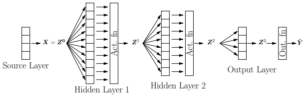

- Exponentially more Compute aka GPUs :-)
- Huge Data
- Deep Neural Networks



- Can Neural Networks solve PDEs ?

What are Deep Neural networks ?



- $\mathcal{L}^*(z) = \sigma_o \odot C_K \odot \sigma \odot C_{K-1} \ldots\ldots\ldots \odot \sigma \odot C_2 \odot \sigma \odot C_1(z)$.
- At the k-th Hidden layer: $z^{k+1} := \sigma(C_k z^k) = \sigma(W_k z^k + B_k)$
- Tuning Parameters: $\theta = \{W_k, B_k\} \in \Theta$,
- σ: scalar Activation function: ReLU, Tanh
- Random Training set: $\mathcal{S} = \{z_i\}_{i=1}^N \in Z$, with i.i.d z_i
- Use SGD (ADAM) to find Target $\mathcal{L} \approx \mathcal{L}^* = \mathcal{L}^*_{\theta^*}$

$$\theta^* := \arg\min_{\theta \in \Theta} \sum_{i=1}^{N} |\mathcal{L}(z_i) - \mathcal{L}^*_\theta(z_i)|^p,$$

Physics Informed Neural Networks

- Variants of PINNs stem from Dissanayake, Phan-Thien, 1994.
- Also in Lagaris et al, mid 1990s.
- Reintroduced by Raissi, Perdikaris, Karniadakis, 2017.
- Extensively developed by Karniadakis and collaborators.
- 10000s of papers on PINNs already.

PINNs for the PDE $\mathcal{D}(\mathrm{u}) = \mathrm{f}$

- For Parameters $\theta \in \Theta$, $\mathrm{u}_\theta : \mathbb{D} \mapsto \mathbb{R}^m$ is a DNN, with $\mathrm{u}_\theta \in X^*$
- Aim: Find $\theta \in \Theta$ such that $\mathrm{u}_\theta \approx \mathrm{u}$ (in suitable sense).
- Compute PDE Residual by Automatic Differentiation:

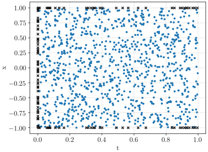
$$\mathcal{R} := \mathcal{R}_\theta(y) = \mathcal{D}\left(\mathrm{u}_\theta(y)\right) - \mathrm{f}(y), \; y \in \mathbb{D} \quad \mathcal{R}_\theta \in Y^*, \quad \forall \theta \in \Theta$$

- PINNs are minimizers of $\|\mathcal{R}_\theta\|_Y^p \sim \int_{\mathbb{D}} |\mathcal{R}_\theta(y)|^p \, dy$
- Replace Integral by Quadrature !
- Let $\mathcal{S} = \{y_i\}_{1 \leq i \leq N}$ be quadrature points in $\mathbb{D}$, with weights w_i
- Could be Random, Sobol, Grid points (Gauss rules)
- PINN for approximating PDE is defined as $\mathrm{u}^* = \mathrm{u}_{\theta^*}$ such that

$$\theta^* = \arg\min_{\theta \in \Theta} \sum_{i=1}^{N} w_i |\mathcal{R}_\theta(y_i)|^p$$

Heat Eqn: $u_t = u_{xx}$ with 0-BC and $u(x,0) = \bar{u}(x)$ IC

- Training Set: $\mathcal{S} = \mathcal{S}_{int} \cup \mathcal{S}_{tb} \cup \mathcal{S}_{sb}$ Randomly chosen.



- Deep Neural networks : $(x,t) \mapsto u_\theta(x,t)$, $\theta \in \Theta$.
- Temporal boundary residual: $\mathcal{R}_{tb,\theta} = u_\theta(\cdot, 0) - \bar{u}$
- Spatial boundary residual: $\mathcal{R}_{sb,\theta} = u_\theta|_{\partial D}$.
- Interior PDE Residual: $\mathcal{R}_{int,\theta} = \partial_t u_\theta - \partial_{xx} u_\theta$
- Evaluate PDE Residual by Automatic Differentiation
- Loss function:

$$J = \frac{1}{N_{tb}} \sum_{n=1}^{N_{tb}} |\mathcal{R}_{tb,\theta}(x_n)|^2 + \frac{1}{N_{sb}} \sum_{n=1}^{N_{sb}} |\mathcal{R}_{sb,\theta}(x_n, t_n)|^2 + \frac{1}{N_{int}} \sum_{n=1}^{N_{int}} |\mathcal{R}_{int,\theta}|^2.$$

Why PINNs are great ?: I

- Very easy to Code !!
- A few lines in PyTorch

```
def compute_res(self, network, x_f_train):
    x_f_train.requires_grad = True
    u = network(x_f_train).reshape(-1, )
    grad_u = torch.autograd.grad(u, x_f_train, grad_outputs=torch.ones(x_f_train.shape[0], ).to(self.device), create_graph=True)[0]
    grad_u_t = grad_u[:, 0]
    grad_u_x = grad_u[:, 1]
    grad_u_xx = torch.autograd.grad(grad_u_x, x_f_train, grad_outputs=torch.ones(x_f_train.shape[0], ).to(self.device), create_graph=True)[0][:, 1

    residual = grad_u_t - self.v * grad_u_xx
    return residual
```

- Don't need Grids !!

Numerical Results: (SM, Molinaro, Tanios, 2021)

- Heat Equation:

Dimension	Training Error	Total error
20	0.006	0.79%
50	0.006	1.5%
100	0.004	2.6%

- Black-Scholes type PDE with Uncorrelated Noise:

Dimension	Training Error	Total error
20	0.0016	1.0%
50	0.0031	1.5%
100	0.0031	1.8%

- Heston option-pricing PDE

Dimension	Training Error	Total error
20	0.0064	1.0%
50	0.0037	1.3%
100	0.0032	1.4%

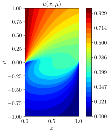
Radiative Transfer Equations

- $2d+1$-dim Integro-Differential PDE for Intensity

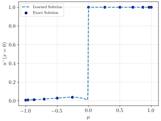
$$\frac{1}{c}u_t + \omega \cdot \nabla u + (k(x,\nu) + \sigma(x,\nu))\, u \\ - \frac{\sigma(x,\nu)}{s_d} \int\limits_{R_+} \int\limits_{S} \Phi(\omega,\omega',\nu,\nu') u d\omega' d\nu' = f(x,t,n,\nu).$$

- High-dimensional, non-local, mixed-type, multiphysics
- PINNs applied and bound derived in SM, Molinaro 2021.

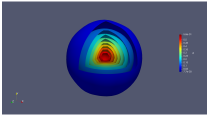
Numerical Results



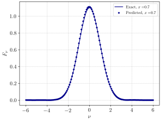
2-D, Intensity



2-D, Boundary



6-D, Inc. Radiation

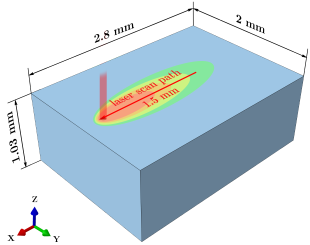


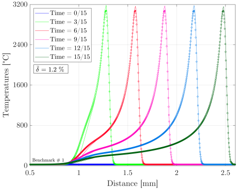
6-D, Radial flux

Dimension	Network Size	Error	Training Time
2	24×8	0.3%	57 min
6	20×8	2.1%	66 min

An Industrial case study

- PINN simulation of Laser Powder Bed Fusion
- Key Component of 3-D Printing





- Traditional FEM Simulation: 4 hrs.
- PINNs of Hosseini et al, 2022: 2×10^{-6} secs with 4% Error.

Why do PINNs work or do they ?

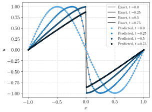
- Based on sound theory.
- Error Bounds of SM, Molinaro, De Ryck et.al 2021-2024:
- For generic PDE: $\mathcal{D}(\mathrm{u}) = f$:

$$\|\mathrm{u} - \mathrm{u}_\theta\| \sim C_{\mathrm{pde}}\left(\mathrm{u}, \mathrm{u}_\theta\right)\left[\mathcal{E}_T(\theta) + C_{\mathrm{quad}}(\mathrm{u}_\theta)N^{-\alpha}\right]$$

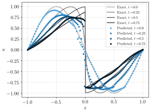
- C_{pde} depends on $\|\nabla \mathrm{u}\|$.
- Can blow up for large gradients.

Viscous Burgers': $u_t + \text{div}\ f(u) = \nu \Delta u$

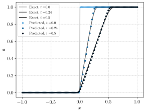
- Error $\mathcal{E} \leq Ce^{CT}\left(\mathcal{E}_T + C_q N^{-\alpha}\right)$, $C = C\left(\|\nabla u^\nu\|_{L^\infty}\right)$
- $\|\nabla u^\nu\|_{L^\infty} \sim \frac{1}{\sqrt{\nu}} \Rightarrow$ Error can blow up near shocks !!



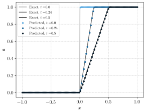
$\nu = 10^{-3}$, Sh



$\nu = 0$, Sh



$\nu = 10^{-3}$, RF



$\nu = 0$, RF

ν	$\mathcal{E}$ (Shock)	$\mathcal{E}$ (Rarefaction)
10^{-3}	1.0%	2.2%
10^{-4}	11.2%	1.6%
0	23.1%	1.2%

What about Training Error ?

- Rigorous Error estimate for PINNs for the PDE $\mathcal{D}(\mathrm{u}) = f$:

$$\|\mathrm{u} - \mathrm{u}_\theta\| \sim C_{\rm pde}\left(\mathrm{u}, \mathrm{u}_\theta\right)\left[\mathcal{E}_T(\theta) + C_{\rm quad}(\mathrm{u}_\theta) N^{-\alpha}\right]$$

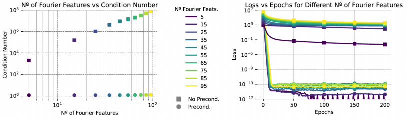
- Training Error is a blackbox
- We have that $\min\limits_{\theta} \mathcal{E}_T(\theta) \leq \epsilon$
- But can we train to reach close to the global minimum with Gradient Descent ?
- De Ryck, SM et al (2024) showed that:

$$N(\delta) \sim \mathcal{O}(\kappa(\mathcal{A}) \log(1/\delta)), \quad \kappa(\mathcal{A}) = \frac{\lambda_{\max}(\mathcal{A})}{\lambda_{\min}(\mathcal{A})}, \quad \mathcal{A} \sim \mathcal{D}^* \mathcal{D}$$

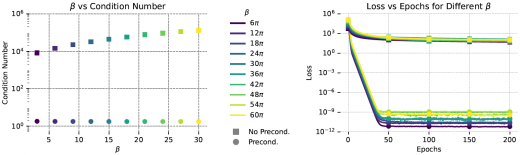
- Convergence of PINNs depends on Conditioning of Hermitian-Square !!
- Ex: if $\mathcal{D} = -\Delta$, then $\mathcal{A} = \Delta^2$

Training PINNs is ill-Conditioned

- For Poisson Equation: $-u'' = -k^2 \sin(kx)$:

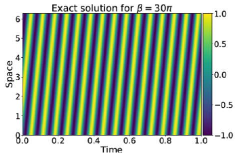


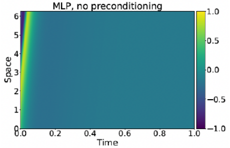
- For Advection Equation: $u_t + \beta u_x = 0$



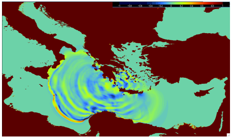
Intrinsic Limitations of PINNs

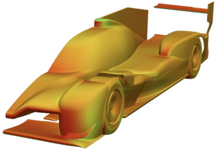
- Don't work on simple problems (Advection with $\beta = 30\pi$)):





- Let alone real use cases !!

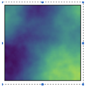




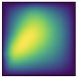
- Preconditioning is an active research area !!
- Have to bring Data to centerstage.

What does solving a PDE entail ?

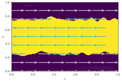
- Finding Solution Operators of PDEs.
- Darcy PDE: $-\mathrm{div}(a\nabla u) = f$, $\mathcal{G} : a \mapsto \mathcal{G}a = u$.



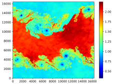
a



u



u_0

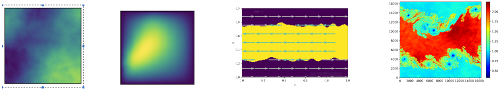


$u(T)$

- Compressible Euler equations: $\mathcal{G} : u_0 \mapsto \mathcal{G}u_0 = u(t)$.
- Operator: $\mathcal{G} : \mathcal{X} \mapsto \mathcal{Y}$, $\dim\,(\mathcal{X}, \mathcal{Y}) = \infty$.
- Learn PDE Solution Operators from Data
- Underlying Data Distribution $\mu \in \mathrm{Prob}\,(\mathcal{X})$
- Draw N i.i.d samples $(a_i, \mathcal{G}(a_i))$ with $a_i \sim \mu$.
- Operator Learning Task: Find approximation to $\mathcal{G}_{\#}\mu$

What does solving a PDE entail ?

- Finding Solution Operators of PDEs.
- Darcy PDE: $-\mathrm{div}(a\nabla u) = f$, $\mathcal{G} : a \mapsto \mathcal{G}a = u$.



a $\quad u \quad u_0 \quad u(T)$

- Compressible Euler equations: $\mathcal{G} : u_0 \mapsto \mathcal{G}u_0 = u(t)$.
- Operator: $\mathcal{G} : \mathcal{X} \mapsto \mathcal{Y}$, $\dim\,(\mathcal{X}, \mathcal{Y}) = \infty$.
- Learn PDE Solution Operators from Data
- Underlying Data Distribution $\mu \in \mathrm{Prob}\,(\mathcal{X})$
- Draw N i.i.d samples $(a_i, \mathcal{G}(a_i))$ with $a_i \sim \mu$.
- Operator Learning Task: Find approximation to $\mathcal{G}_{\#}\mu$
- Caveat: Neural Networks: $\mathbb{R}^N \mapsto \mathbb{R}^M$

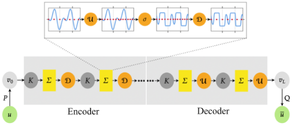
Possible Solution: Neural Operators

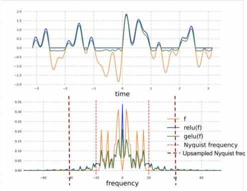
- DNNs are $\mathcal{L}_\theta = \sigma_K \odot \sigma_{K-1} \odot \ldots\ldots \odot \sigma_1$
- Single hidden layer: $\sigma_k(y) = \sigma(A_k y + B_k)$, with $y \in \mathbb{R}^N$
- Generalize DNNs to ∞-dimensions: Kovachki et al, 2021:
- NO: $\mathcal{N}_\theta = \mathcal{N}_L \odot \mathcal{N}_{L-1} \odot \ldots\ldots \odot \mathcal{N}_1$
- Single hidden layer;

$$(\mathcal{N}_\ell v)(x) = [\mathcal{P}\sigma] \left(B_\ell(x) + \int_D K_\ell(x,y) v(y) dy \right)$$

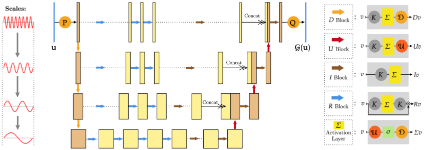
- Kernel Integral Operators with Parameters B_ℓ, K_ℓ
- Nonlocal activations $[\mathcal{P}\sigma]$ Bartolucci, SM et. al, 2023

Convolutional Neural Operators: Raonic, SM et al, 2023



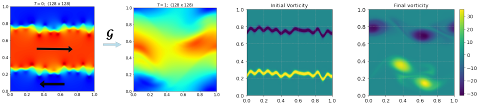


- I: Use Convolutional Kernels in Physical space
- II: Modulated Nonlocal activations for Alias-free processing.
- CNO instantiated as a modified Operator UNet

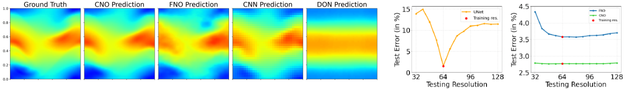


Example: Navier-Stokes Eqns.

- Operator:



- Comparison:



- Test Errors:

Model	FFNN	UNet	DeepONet	FNO	CNO
Error	8.05%	3.54%	11.64%	3.93%	3.01%

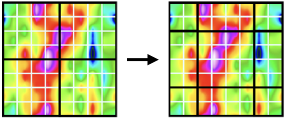
- CNO is Resolution Invariant a la Bartolucci, SM et. al, 2023

What about Nonlinear Kernels ?

- Operator Attention: $\mathbb{A}(\mathrm{v})(x) = \int\limits_D K(v(x), v(y))v(y)dy$:

$$\mathrm{u}(x) = \mathbb{A}(\mathrm{v})(x) = \mathrm{W} \int\limits_D \frac{e^{\frac{\langle \mathrm{Qv}(x), \mathrm{Kv}(y) \rangle}{\sqrt{m}}}}{\int\limits_D e^{\frac{\langle \mathrm{Qv}(z), \mathrm{Kv}(y) \rangle}{\sqrt{m}}} dz} \mathrm{Vv}(y)dy.$$

- Computational Cost is Quadratic in # (Tokens) !!
- Scaling through Vision + SWIN transformers



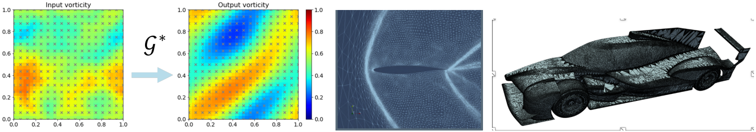
- scOT of Herde,SM et. al., 2024.

Models perform very well on 2-D Cartesian Domains !!

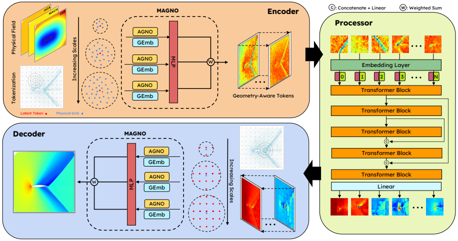
- Extensive Empirical evaluation on RPB benchmarks.

	In/Out	FFNN	GT	UNet	ResNet	DON	FNO	CNO
Poisson Equation	In	5.74%	2.77%	0.71%	0.43%	12.92%	4.98%	**0.21%**
	Out	5.35%	2.84%	1.27%	1.10%	9.15%	7.05%	**0.27%**
Wave Equation	In	2.51%	1,44%	1.51%	0.79%	2.26%	1.02%	**0.63%**
	Out	3.01%	1.79%	2.03%	1.36%	2.83%	1.77%	**1.17%**
Smooth Transport	In	7.09%	0.98%	0.49%	0.39%	1.14%	0.28%	**0.24%**
	Out	650.6%	875.4%	1.28%	0.96%	157.2%	3.90%	**0.46%**
Discontinuous Transport	In	13.0%	1.55%	1.31%	**1.01%**	5.78%	1.15%	**1.01%**
	Out	257.3%	22691.1%	1.35%	1.16%	117.1%	2.89%	**1.09%**
Allen-Cahn Equation	In	18.27%	0.77%	0.82%	1.40%	13.63%	**0.28%**	0.54%
	Out	46.93%	2.90%	2.18%	3.74%	19.86%	**1.10%**	2.23%
Navier-Stokes Equations	In	8.05%	4.14%	3.54%	3.69%	11.64%	3.57%	**2.76%**
	Out	16.12%	11.09%	10.93%	9.68%	15.05%	9.58%	**7.04%**
Darcy Flow	In	2.14%	0.86%	0.54%	0.42%	1.13%	0.80%	**0.38%**
	Out	2.23%	1.17%	0.64%	0.60%	1.61%	1.11%	**0.50%**
Compressible Euler	In	0.78%	2.09%	0.38%	1.70%	1.93%	0.44%	**0.35%**
	Out	1.34%	2.94%	0.76%	2.06%	2.88%	0.69%	**0.59%**

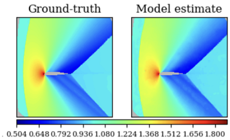
Caveat I: PDEs on Arbitrary Domains

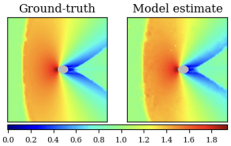


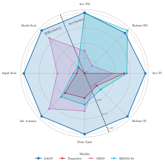
- Discussion so far has only focussed on Cartesian Domains
- Discretized with Uniform Grids.
- Most Real world PDEs are on Arbitrary Domains
- Discretized with Unstructured Grids or Point Clouds
- Need to handle such Data !!



- Geometry Aware Operator Transformer: Wen, SM et. al, 2025
- GAOT is both accurate and efficient.

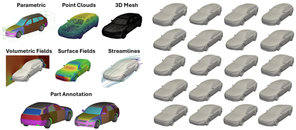




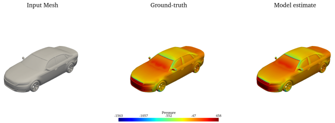


The DrivaerNet++ Challenge

- Flow past Cars Dataset (8K Car Shapes with 2M nodes each)



- GAOT: SOTA for Surface Pressure, Shear Stress

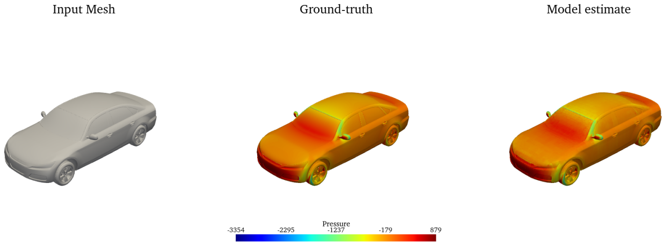


Model	GAOT	FigConvNet	TripNet	RegDGCNN
L^1 Pressure Err.	0.110	0.122	0.125	0.161
L^1 Shear Err.	0.156	0.222	0.215	0.364

- CFD: 300 Node hours vs. GAOT: 0.36 seconds !!!

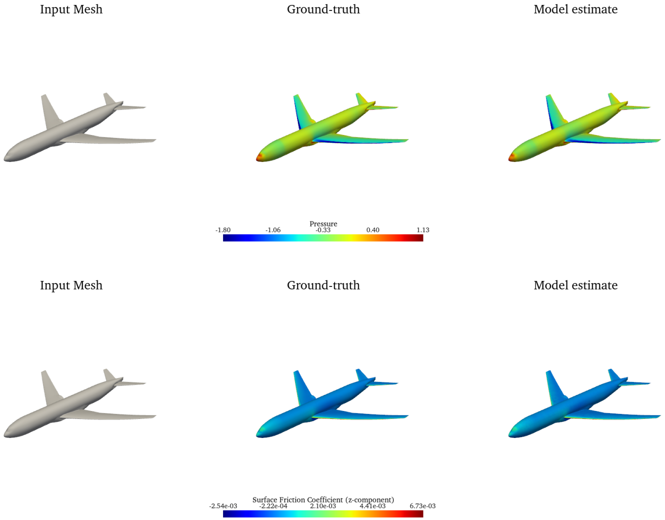
The DriverML challenge

- HR-LES simulations of flow past 500 cars.
- More accurate than RANS for Drivearnet++.
- Upto 10 M surface nodes handled accurately by GAOT !!



Flow Past an entire Aircraft

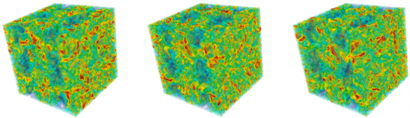
- AIAA's NASA CRM Benchmark.
- GAOT predicts Surface Pressure+Skin Friction accurately !!



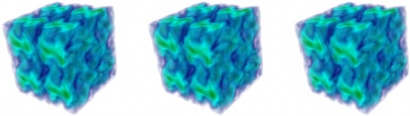
Caveat II: PDE with Chaotic Multiscale Solutions

- 3-D Navier-Stokes with Taylor-Green initial data.

- Spectral Viscosity Method:



- Convolutional Fourier Neural Operator (C-FNO):



- All ML models trained to minimize MSE or MAE:
  - Smooth out Small Scales
  - Collapse to Mean

Why does this happen ?: Molinaro, SM et. al, 2025

- Insensitivity of Neural Networks:

$$\Psi_\theta(u + \delta u) \approx \Psi_\theta(u), \ \delta u << 1$$

- DNNs are optimal at the Edge of Chaos: $\mathrm{Lip}(\Psi_\theta) \sim \mathcal{O}(1)$
- Spectral Bias of DNNs
- Bounded Gradients are essential for training with GD
- Implication $\Rightarrow$ DNNs will Collapse to Mean !!

$$\mathbb{E}_{\delta\bar{u}} \|\Psi_\theta(\bar{u}^* + \delta\bar{u}) - \mathcal{S}(\bar{u}^* + \delta\bar{u})\|^2 \approx \mathbb{E}_{\delta\bar{u}} \|\Psi_\theta(\bar{u}^*) - \mathcal{S}(\bar{u}^* + \delta\bar{u})\|^2 \quad \text{(insensitivity hypothesis)}$$
$$= \|\Psi_\theta(\bar{u}^*) - \mathbb{E}_{\delta\bar{u}} \mathcal{S}(\bar{u}^* + \delta\bar{u})\|^2 + \mathrm{Var}_{\delta\bar{u}}[\mathcal{S}(\bar{u}^* + \delta\bar{u})]. \quad \text{(bias-variance decomposition)}$$

Why does this happen ?: Molinaro, SM et. al, 2025

- Insensitivity of Neural Networks:

$$\Psi_\theta(u + \delta u) \approx \Psi_\theta(u), \ \delta u << 1$$

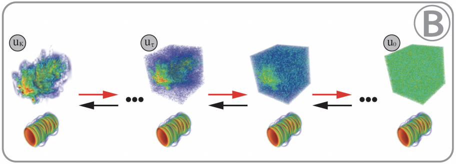
- DNNs are optimal at the Edge of Chaos: $\mathrm{Lip}(\Psi_\theta) \sim \mathcal{O}(1)$
- Spectral Bias of DNNs
- Bounded Gradients are essential for training with GD
- Implication $\Rightarrow$ DNNs will Collapse to Mean !!

$$\mathbb{E}_{\delta\bar{u}} \|\Psi_\theta(\bar{u}^* + \delta\bar{u}) - \mathcal{S}(\bar{u}^* + \delta\bar{u})\|^2 \approx \mathbb{E}_{\delta\bar{u}} \|\Psi_\theta(\bar{u}^*) - \mathcal{S}(\bar{u}^* + \delta\bar{u})\|^2 \quad \text{(insensitivity hypothesis)}$$
$$= \|\Psi_\theta(\bar{u}^*) - \mathbb{E}_{\delta\bar{u}} \mathcal{S}(\bar{u}^* + \delta\bar{u})\|^2 + \mathrm{Var}_{\delta\bar{u}}[\mathcal{S}(\bar{u}^* + \delta\bar{u})]. \quad \text{(bias-variance decomposition)}$$

- Directly Learn the Conditional Distribution $\mathcal{S}_{\#}\mu$

GenCFD algorithm of Molinaro et. al, SM, 2025

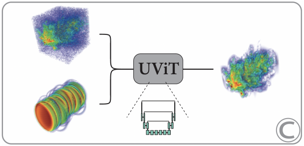
- Based on Conditional Score Based Diffusion Models



- Denoised with the Reverse SDE:

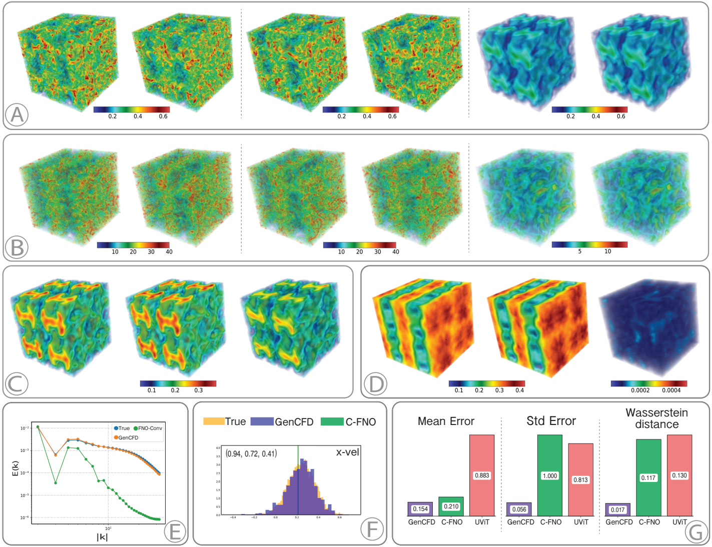
$$du_\tau = 2\left(\frac{\dot{\sigma}_\tau}{\sigma_\tau} + \frac{\dot{s}_\tau}{s_\tau}\right)d\tau - 2s_\tau\frac{\dot{\sigma}_\tau}{\sigma_\tau}D_\theta(\Delta t,\, u_{\tau+1},\, \overline{u},\, \sigma_\tau)d\tau + s\sqrt{2\dot{\sigma}_\tau\sigma_\tau}\, d\widehat{W}_\tau$$

- Denoiser minimizes $\mathbb{E}\|u(t_n, \bar{u}) - D_\theta(u(t_n, \bar{u}) + \eta, \bar{u}, \sigma)\|$

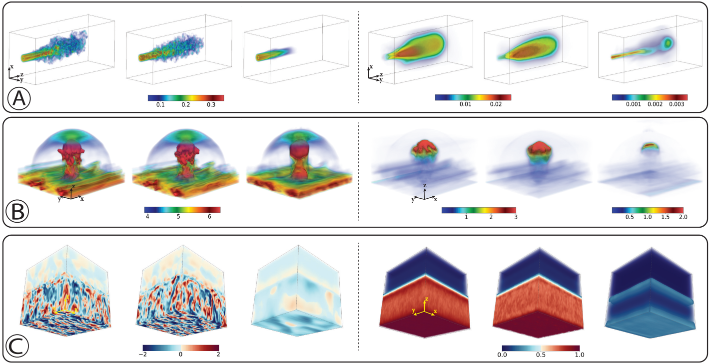


- GenCFD provably approximates the Conditional Distribution !!

Taylor-Green Results

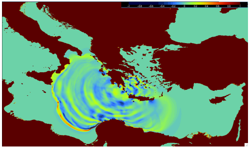


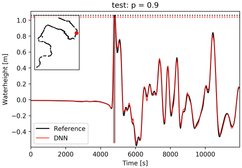
GenCFD works very well for Realworld Flows

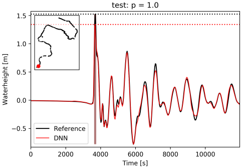


- Nozzle Jet: 3.5 hrs (LBM) vs. GenCFD: 1.45s
- Cloud-Shock: 5 hrs (FVM) vs. GenCFD: 0.45s
- Conv. Boundary Layer: 13.3 hrs (FDM) vs. GenCFD: 3.8s

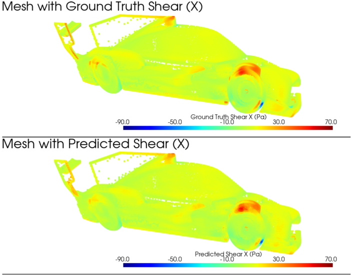
What about the Use Cases ?



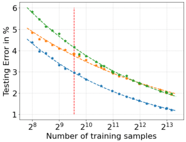




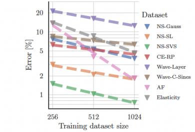
- AI Tsunami Simulation takes 10^{-3} secs (vs. 1 hr)
- AI RaceCar Simulation takes 10^{-2} secs (vs. 500 Node hrs)



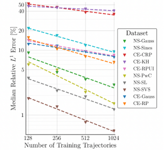
Where's the CAVEAT ?



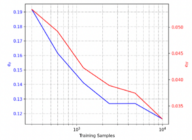
CNO/FNO



RIGNO



GAOT

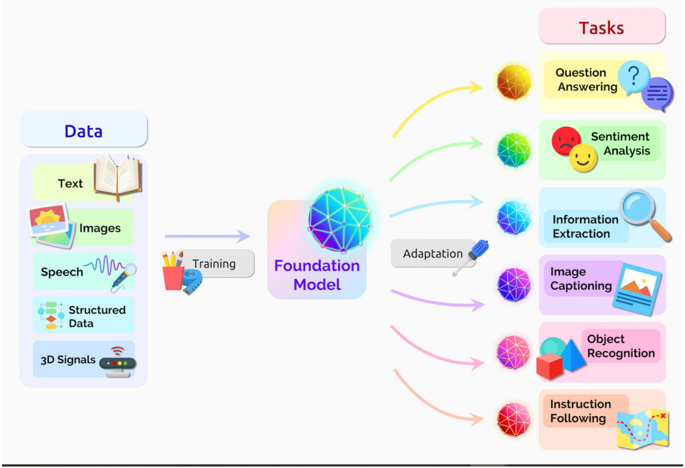


GenCFD

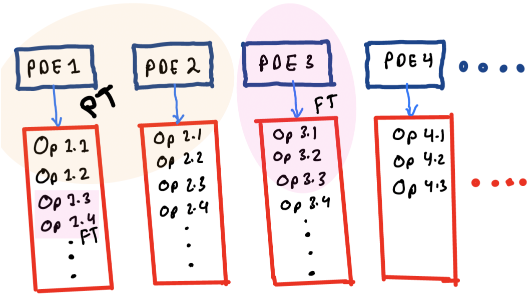
- Models Scale with sample size: $\mathcal{E} \sim N^{-\alpha}$ but with α small
- Even more pessimistic rates with theory [1]
- ML models require Big Data: $\mathcal{O}(10^3) - \mathcal{O}(10^4)$ training samples per Task
- Very Difficult to obtain Data for PDEs.
- How to make models much more Sample Efficient ?

[1] Lanthaler, SM, Karniadakis, 2022, De Ryck, SM, 2023

Foundation Models are the Key for Text/Vision!!

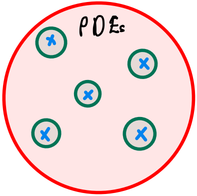


What would a Foundation Model for PDEs look like ?

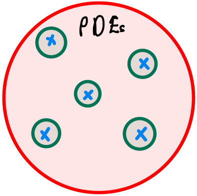


- Op: Operators need PDE + Data.
- PT: Pretraining.
- FT: Finetuning (Adaptation)

Can it Work ?

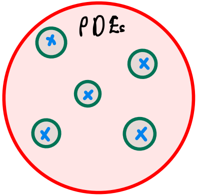


Can it Work ?



- Lets try it out !!

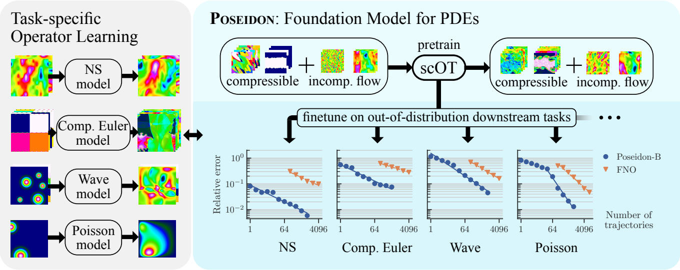
Can it Work ?



- Lets try it out !!

POSEIDON

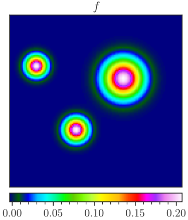
- PDE foundation model of Herde, SM et. al, 2024
- Pretrained on Euler + Navier-Stokes Eqns in 2-D.
- Finetuned on 15 Unseen Tasks
- Paper + Code: https://github.com/camlab-ethz/poseidon
- Model + Datasets: https://huggingface.co/camlab-ethz



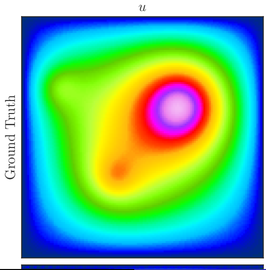
Summary of Performance on all Downstream Tasks

Deep Dive: Poisson Eqn.

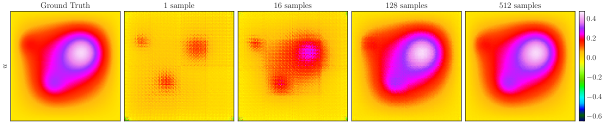
- Input (Source):



- Output (Solution):

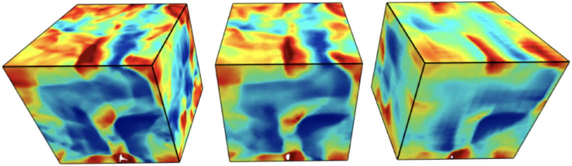


How much Physics has been learnt in PreTraining ?



Ongoing: Scaling up POSEiDON

- Increase Model Size by $4x$: 2.5B Model.
- Increase Dataset Size by $10 - 50$ x
- Augment Model Features:
  - 3D
  - Unstructured (point cloud) inputs
  - Genuine Multiphysics Training
  - Diffusion model for multiscale problems
  - PDE symbolic information



Summary+ Outlook

- ML/AI model can be potential Neural PDE Solvers (PINNs)
- Training is intrinsically Ill-conditioned.
- ML/AI models are effective Neural PDE Surrogates:
  - Neural Operators (CNO, scOT) for PDEs on Cartesian Domains.
  - Graphs+Transformers (GAOT) for Arbitrary Domains.
  - Diffusion Models (GenCFD) for Multiscale, Chaotic PDEs.
- Sample Efficiency is the main challenge.
- Foundation Models (Poseidon) can address it.
- They need to be Scaled Up significantly.